

First-principles multiway spectral partitioning of graphs

Maria A. Riolo^{1,2} and M. E. J. Newman^{2,3}

¹*Department of Mathematics, University of Michigan, Ann Arbor, MI 48109*

²*Center for the Study of Complex Systems, University of Michigan, Ann Arbor, MI 48109*

³*Department of Physics, University of Michigan, Ann Arbor, MI 48109*

We consider the minimum-cut partitioning of a graph into more than two parts using spectral methods. While there exist well-established spectral algorithms for this problem that give good results, they have traditionally not been well motivated. Rather than being derived from first principles by minimizing graph cuts, they are typically presented without direct derivation and then proved after the fact to work. In this paper, we take a contrasting first-principles approach in which we start with a matrix formulation of the minimum cut problem and then show, via a relaxed optimization, how it can be mapped onto a spectral embedding defined by the leading eigenvectors of the graph Laplacian. The end result is an algorithm that is similar in spirit to, but different in detail from, previous spectral partitioning approaches. In tests of the algorithm we find that it outperforms previous approaches on certain particularly difficult partitioning problems.

I. INTRODUCTION

Graph partitioning, the division of a graph or network into weakly connected subgraphs, is a problem that arises in many areas, including finite element analysis, electronic circuit design, parallel computing, network data analysis and visualization, and others [1, 2]. In the most basic formulation of the problem one is given an undirected, unweighted graph of n vertices and asked to divide the vertices into k nonoverlapping groups of given sizes, such that the number of edges running between groups—the so-called *cut size*—is minimized. This is known to be a computationally hard problem. Even for the simplest case where $k = 2$ it is NP-hard to find the division with minimum cut size [3]. Good approximations to the minimum can, however, be found using a variety of heuristic methods, including local greedy algorithms, genetic algorithms, tabu search, and multilevel algorithms. One particularly elegant and effective approach, which is the subject of this paper, is *spectral partitioning* [4], which makes use of the spectral properties of any of several matrix representations of the graph, most commonly the graph Laplacian.

The first spectral partitioning algorithms date back to the work of Fiedler in the 1970s [5, 6] and were aimed at solving the graph bisection problem, i.e., the problem of partitioning a graph into just two parts. For this problem the underlying theory of the spectral method is well understood and the algorithms work well. One calculates the eigenvector corresponding to the second lowest eigenvalue of the graph Laplacian and then divides the graph according to the values of the vector elements—the complete process is described in Section II. More recently, attention has turned to the general multiway partitioning problem with arbitrary k , which is harder. One simple approach is to repeatedly bisect the graph into smaller and smaller parts using Fiedler’s method or one of its variants, but this can give rise to poor solutions in some commonly occurring situations. A better approach, and the one in widest current use, is to construct the $n \times (k-1)$

matrix whose columns are the eigenvectors corresponding to the second- to k th-lowest eigenvalues of the Laplacian. The rows of this matrix then define n vectors of $k - 1$ elements each which are regarded as points in a $(k - 1)$ -dimensional space. One clusters these points into k groups using any of a variety of heuristics, the most common being the k -means method, and the resulting clusters define the division of the graph.

As is common in the computer science literature, the presentation of algorithms in this class is not normally in the form of an explicit derivation of the algorithm (though there is the occasional exception, such as [7]). Instead, the algorithm is proposed without justification, and justified after the fact by demonstrating that it performs well on particular partitioning tasks, or that it provides certain performance guarantees. Philosophically the approach is similar to that of the mathematician who demonstrates a result by first conjuring a theorem out of thin air and then afterwards proving it to be correct.

While this approach is perfectly logical and technically correct, it does not typically give us much understanding of an algorithm. For that, it would be better to derive the algorithm from first principles. The purpose of this paper is to give such a derivation for multiway spectral partitioning. Our main goal in doing so is to gain an understanding of *why* spectral partitioning works, by contrast with the traditional presentation which demonstrates only that it does work. However, as we will see, the algorithm that we derive in the process is not, in fact, identical to previous spectral partitioning algorithms and in Section IV we present results that indicate that our algorithm can outperform more conventional approaches for certain classes of graphs.

The previous literature on spectral partitioning is extensive—this has been an active area of research, especially in the last few years. There is also a large literature on the related problem of spectral clustering of high-dimensional data sets, which can be mapped onto a weighted partitioning problem on a complete graph using an affinity matrix. The 1995 paper of Alpert and Yao [7] provides an early example of an explicit derivation of

a general multiway partitioning algorithm. Their algorithm is substantially different from the most commonly used variants, involving a vector partitioning step, and also includes one arbitrary parameter which affects the performance of the algorithm but whose optimal value is unknown. The algorithms of Shi and Malik [8] and Meilă and Shi [9] are good examples of the standard multiway partitioning using k -means, although applied to the slightly different problem of normalized-cut partitioning. A number of subsequent papers have analyzed these algorithms or variants of them [10–13]. Summaries are given by von Luxburg [4], Verma and Meilă [14], and Bach and Jordan [15], although the discussions are in the language of data clustering, not graph partitioning. Perhaps the work that comes closest to our own is that of Zhang and Jordan [16], again on data clustering, in which partitions are indexed using a set of $(k - 1)$ -dimensional “margin vectors,” which are oriented using a Procrustes technique, as we also do, although other details of their approach are different from ours.

The outline of this paper is as follows. In Section II we review the derivation of the standard spectral bisection algorithm and then in Section III present in detail the generalization of that derivation to the multiway partitioning problem, leading to a straightforward algorithm for multiway partitioning of an arbitrary undirected graph. In Section IV we give example applications of this algorithm to a number of test graphs, and demonstrate that its performance is similar to, or in some cases slightly better than, approaches based on k -means. In Section V we give our conclusions and discuss directions for future research.

II. SPECTRAL BISECTION

The term *spectral bisection* refers to the special case in which we partition a graph into exactly $k = 2$ parts. For this case there is a well-established first-principles derivation of the standard partitioning algorithm, which we review in this section. Our goal in subsequent sections will be to find a generalization to the case of arbitrary k .

Suppose we are given an undirected, unweighted graph on n vertices, which we will assume to be connected (i.e., to have only one component), and we wish to divide the vertices into two groups which, for the sake of simplicity, we will take to be of equal size $\frac{1}{2}n$ (with n even). We define an index variable s_i for each vertex $i = 1 \dots n$ such that $s_i = 1$ if vertex i belongs to group 1 and $s_i = -1$ if i belongs to group 2. We note that

$$\frac{1}{2}(s_i s_j + 1) = \begin{cases} 1 & \text{if } i, j \text{ are in the same group,} \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Thus the number of edges within groups is given by $\frac{1}{2} \sum_{ij} \frac{1}{2}(s_i s_j + 1) A_{ij}$, where A_{ij} is an element of the adjacency matrix (having value 1 if there is an edge between i and j , and zero otherwise) and the extra factor of $\frac{1}{2}$ com-

pensates for double counting of vertex pairs in the sum. The total number of edges in the entire graph is $\frac{1}{2} \sum_{ij} A_{ij}$ and hence the number of edges between groups—which is the cut size R —is given by

$$\begin{aligned} R &= \frac{1}{2} \sum_{ij} A_{ij} - \frac{1}{4} \sum_{ij} (s_i s_j + 1) A_{ij} \\ &= \frac{1}{4} \sum_{ij} (1 - s_i s_j) A_{ij} = \frac{1}{4} \sum_i d_i - \frac{1}{4} \sum_{ij} s_i s_j A_{ij}, \end{aligned} \quad (2)$$

where $d_i = \sum_j A_{ij}$ is the degree of vertex i . Noting that $s_i^2 = 1$ for all i , this equation can be rewritten as

$$R = \frac{1}{4} \sum_{ij} d_i \delta_{ij} s_i s_j - \frac{1}{4} \sum_{ij} A_{ij} s_i s_j = \frac{1}{4} \sum_{ij} L_{ij} s_i s_j, \quad (3)$$

where $L_{ij} = d_i \delta_{ij} - A_{ij}$ is the ij th element of the graph Laplacian matrix \mathbf{L} . Alternatively, we can write R in matrix notation as

$$R = \frac{1}{4} \mathbf{s}^T \mathbf{L} \mathbf{s}, \quad (4)$$

where \mathbf{s} is the n -component vector with elements s_i .

Our goal, for a given graph and hence for given \mathbf{L} , is to minimize the cut size R over possible bisections of the graph, represented by \mathbf{s} , subject to the constraint that the two groups are the same size, which is equivalent to saying that $\sum_i s_i = 0$ or

$$\mathbf{1}^T \mathbf{s} = 0, \quad (5)$$

where $\mathbf{1} = (1, 1, 1, \dots)$. Unfortunately, as mentioned in the introduction, this is a hard computational problem. But one can in many cases find reasonably good solutions in polynomial time by using a *relaxation method*. We generalize the discrete variables $s_i = \pm 1$ to continuous real variables x_i and solve the relaxed minimization with respect to the vector $\mathbf{x} = (x_1, x_2, \dots)$ of

$$R_x = \frac{1}{4} \mathbf{x}^T \mathbf{L} \mathbf{x}, \quad (6)$$

subject to the constraint

$$\mathbf{1}^T \mathbf{x} = 0, \quad (7)$$

which is the equivalent of Eq. (5). One must however also apply an additional constraint to prevent \mathbf{x} from becoming zero, which is normally taken to have the form

$$\mathbf{x}^T \mathbf{x} = n. \quad (8)$$

Choices of \mathbf{x} satisfying this second constraint include all allowed values of the original unrelaxed vector \mathbf{s} , since $\mathbf{s}^T \mathbf{s} = \sum_i s_i^2 = \sum_i 1 = n$, but also include many other values in addition. Geometrically, one can think of \mathbf{s} as defining a point in an n -dimensional space, with the allowed values $s_i = \pm 1$ restricting the point to fall at one of the corners of a hypercube. The value of \mathbf{x} falls on the circumscribing hypersphere, since $\mathbf{x}^T \mathbf{x} = \sum_i x_i^2 = n$ implies that \mathbf{x} has constant length \sqrt{n} . The hypersphere

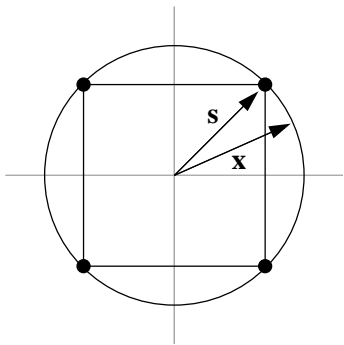


FIG. 1: The possible values of the vector \mathbf{s} lie at the corners of a hypercube in n -dimensional space, while the vector \mathbf{x} can lie at any point on the circumscribing hypersphere.

coincides with the values of \mathbf{s} at the corners of the hypercube, but includes other values in between as well—see Fig. 1.

The relaxed problem is straightforward to solve by differentiation. We enforce the two conditions (7) and (8) with Lagrange multipliers λ and μ so that

$$\frac{\partial}{\partial x_k} \left[\sum_{ij} L_{ij} x_i x_j - \lambda \sum_i x_i^2 - \mu \sum_i x_i \right] = 0. \quad (9)$$

Performing the derivatives we find that $2 \sum_j L_{kj} x_j - 2\lambda x_k - \mu = 0$ or in matrix notation $\mathbf{L}\mathbf{x} = \lambda\mathbf{x} + \frac{1}{2}\mu\mathbf{1}$. Multiplying on the left by $\mathbf{1}$ we get $\mathbf{1}^T \mathbf{L}\mathbf{x} = \lambda \mathbf{1}^T \mathbf{x} + \frac{1}{2}n\mu$, and employing Eq. (7) and noting that $\mathbf{1}$ is an eigenvalue of \mathbf{L} with eigenvalue zero, we find that $\mu = 0$. Thus we end up with

$$\mathbf{L}\mathbf{x} = \lambda\mathbf{x}. \quad (10)$$

In other words \mathbf{x} is an eigenvector of the graph Laplacian satisfying the two conditions (7) and (8).

Our solution is completed by noting that the cut size R_x within the relaxed approximation, evaluated at the solution of (10), is

$$R_x = \frac{1}{4} \mathbf{x}^T \mathbf{L}\mathbf{x} = \frac{1}{4} \lambda \mathbf{x}^T \mathbf{x} = \frac{n\lambda}{4}. \quad (11)$$

This is minimized by choosing λ as small as possible, in other words by choosing \mathbf{x} to be the eigenvector corresponding to the lowest possible eigenvalue. The lowest eigenvalue of \mathbf{L} is always zero, with corresponding eigenvector proportional to $\mathbf{1}$, but we cannot choose this eigenvector because it is forbidden by the condition $\mathbf{1}^T \mathbf{x} = 0$, which requires that the solution vector \mathbf{x} be orthogonal to $\mathbf{1}$. (This is equivalent to saying that we're not allowed to put all vertices in the same one group, which would certainly ensure a small cut size, but wouldn't give a bisection of the graph.) Our next best choice is to choose \mathbf{x} proportional to the eigenvector for the second-lowest eigenvalue, the so-called *Fiedler vector*.

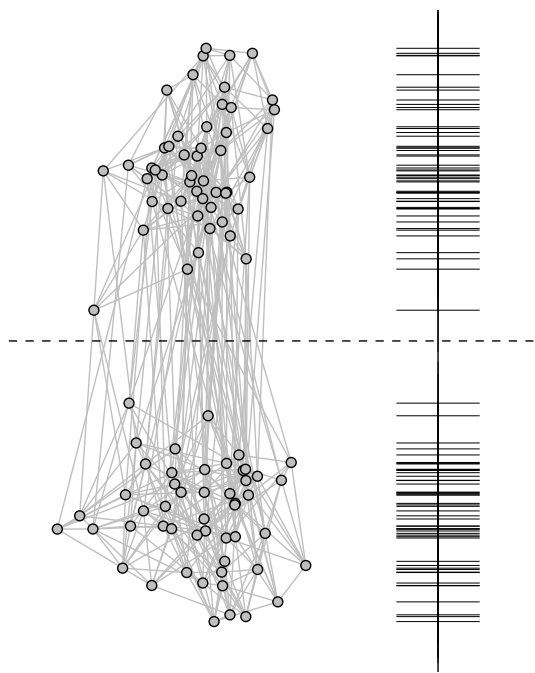


FIG. 2: Left: a small, computer-generated graph with two equally sized groups of vertices. Right: the elements of the Fiedler vector—the second eigenvector of the graph Laplacian—plotted on an arbitrary scale. A division of the vertices into two groups according to the signs of these elements (the dashed line indicates zero) recovers the groups.

This solves the relaxed optimization problem exactly. The final step in the process is to “unrelax” back to the original variables $s_i = \pm 1$, which we do by rounding the x_i to the nearest value ± 1 , which means that positive values of x_i are rounded to $+1$ and negative values to -1 . Thus our final algorithm is a straightforward one: we calculate the eigenvector of the graph Laplacian corresponding to the second-lowest eigenvalue and then divide the vertices into two groups according to the signs of the elements of this vector. Although the solution of the relaxed optimization is exact, the unrelaxation process is only an approximation—there is no guarantee that rounding to ± 1 gives the correct optimum for the unrelaxed problem—and hence the overall algorithm only gives an approximate solution to the partitioning problem. In practice, however, it appears to work well. Figure 2 shows an example.

As we have described it, the algorithm above also does not guarantee that the two final groups of vertices will be equal in size. The relaxed optimization guarantees that $\sum_i x_i = 0$ because of Eq. (7), but we are not guaranteed that $\sum_i s_i = 0$ after the rounding procedure. Normally $\sum_i s_i$ will be close to zero, and hence the groups will be of nearly equal sizes, but there can be some imbalance. In typical usage, however, this is not a problem. In many applications one is willing to put up with a small imbalance anyway, but if one is not then a post-processing step

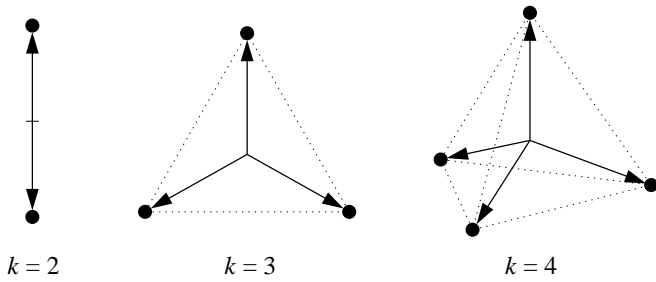


FIG. 3: As group labels we use vectors \mathbf{w}_i pointing from the center to the corners of a regular $(k-1)$ -dimensional simplex. For $k=2$ the simplex consists of just two points on a line, consistent with the indices $s_i = \pm 1$ used in Section II. For $k=3$ the simplex is an equilateral triangle; for $k=4$ it is a regular tetrahedron. In higher dimensions it takes the form of the appropriate generalization of a tetrahedron to four or more dimensions, which would be difficult to draw on this two-dimensional page.

can be performed that moves a small number of vertices between groups in order to restore balance.

III. GENERALIZATION TO MORE THAN TWO GROUPS

Our primary purpose in this paper is to give a generalization of the derivation of the previous section for spectral partitioning into more than two groups. As we will see, a true generalization leads to an algorithm that differs in significant ways from previous multiway partitioning algorithms.

A. Cut size for multiway partitioning

To generalize the spectral bisection algorithm to the case of more than two groups we need first to find an appropriate generalization of the quantities s_i used in Section II to denote membership of the different communities. For a partitioning into k groups, we propose using vectors \mathbf{w}_i that point to the k vertices of a $(k-1)$ -dimensional regular simplex centered on the origin. For $k=3$, for example, the three vectors would point to the corners of an equilateral triangle; for $k=4$ the vectors would point to the corners of a regular tetrahedron, and so forth—see Fig. 3.

Such simplex vectors are not orthogonal. Rather they satisfy a relation of the form

$$\mathbf{w}_i^T \mathbf{w}_j = \delta_{ij} - \frac{1}{k}. \quad (12)$$

Note that the individual vectors are normalized so that $\mathbf{w}_i^T \mathbf{w}_i = 1 - 1/k$. (One could normalize them to have unit length, but subsequent formulas work out less neatly that way.) One can also consider the k simplex vectors to be

the rows of a $k \times (k-1)$ matrix \mathbf{W} , in which case Eq. (12) is equivalent to $\mathbf{W}\mathbf{W}^T = \mathbf{I} - \mathbf{1}\mathbf{1}^T/k$, where \mathbf{I} is the identity and $\mathbf{1} = (1, 1, 1, \dots)$ as previously. The matrix also satisfies an orthogonality condition on its columns of the form

$$\mathbf{W}^T \mathbf{W} = \mathbf{I}. \quad (13)$$

We make use of these simplex vectors in the partitioning problem by assigning one vector to represent each of the groups. All assignments are equivalent, up to rotations and reflections, and any choice will work equally well. We label each vertex i with a vector variable \mathbf{s}_i equal to the simplex vector for its group. Then Eq. (12) implies that

$$\mathbf{s}_i^T \mathbf{s}_j + \frac{1}{k} = \begin{cases} 1 & \text{if } i, j \text{ are in the same group,} \\ 0 & \text{otherwise,} \end{cases} \quad (14)$$

which is the equivalent of Eq. (1), and the derivation of the cut size follows through as before, giving

$$R = \frac{1}{2} \sum_{ij} (d_i \delta_{ij} - A_{ij}) \mathbf{s}_i^T \mathbf{s}_j = \frac{1}{2} \sum_{ij} L_{ij} \mathbf{s}_i^T \mathbf{s}_j, \quad (15)$$

where L_{ij} is once again an element of the graph Laplacian matrix \mathbf{L} . Alternatively, we can introduce an $n \times (k-1)$ matrix \mathbf{S} whose i th row is equal to \mathbf{s}_i and write the cut size in matrix notation as

$$R = \frac{1}{2} \text{Tr}(\mathbf{S}^T \mathbf{L} \mathbf{S}). \quad (16)$$

Since the vertices of the regular simplex are all equivalent, the vectors pointing to them necessarily sum to zero $\sum_i \mathbf{w}_i = 0$. Assuming once again that we want to divide the graph into equally sized groups, we can enforce this equality by requiring that $\sum_i \mathbf{s}_i = 0$, or equivalently

$$\mathbf{1}^T \mathbf{S} = 0. \quad (17)$$

For equally sized groups Eq. (13) implies that the matrix \mathbf{S} also automatically satisfies the condition

$$\mathbf{S}^T \mathbf{S} = \frac{n}{k} \mathbf{I}, \quad (18)$$

which is the equivalent of the condition that $\sum_i s_i^2 = n$ in the two-group case.

B. Minimization of the cut size

Our goal is to minimize the cut size (16) subject to the constraint (17). Once again this is a hard computational problem, but, by analogy with the two-group case, we can render it tractable by relaxing the requirement that each row of \mathbf{S} be equal to one of the discrete simplex vectors, solving this relaxed problem exactly, then rounding to the simplex vectors again to get an approximate solution to the original unrelaxed problem.

Specifically, we replace \mathbf{S} with a matrix \mathbf{X} of continuous-valued elements to give a relaxed cut size

$$R_x = \frac{1}{2} \text{Tr}(\mathbf{X}^T \mathbf{L} \mathbf{X}), \quad (19)$$

where the elements of \mathbf{X} will be allowed to take any real value subject to the constraint

$$\mathbf{1}^T \mathbf{X} = 0, \quad (20)$$

equivalent to Eq. (17). Again, however, we also need an additional constraint, equivalent to Eq. (8), to prevent all elements of \mathbf{X} from becoming zero (which would certainly minimize R_x , but would not give a useful partition of the graph). The natural choice for this additional constraint is the generalization of (18):

$$\mathbf{X}^T \mathbf{X} = \frac{n}{k} \mathbf{I}. \quad (21)$$

As in the two-group case, choices of \mathbf{X} satisfying this condition necessarily include as a subset the original simplex vectors that satisfy (18), but also include many other choices as well. Between them, the two conditions (20) and (21) imply that the columns of \mathbf{X} should be identically normalized, orthogonal to one another, and orthogonal to the vector $\mathbf{1}$. As we now show, the correct choice that satisfies all of these conditions and minimizes R_x is to make the columns proportional to the eigenvectors of the graph Laplacian corresponding to the second- to k th-lowest eigenvalues.

The relaxed cut size (19) can be minimized, as before, by differentiating, applying the conditions (20) and (21) with Lagrange multipliers:

$$\begin{aligned} \frac{\partial}{\partial X_{kl}} \left[\sum_{ijm} L_{ij} X_{im} X_{jm} \right. \\ \left. - \sum_{imn} \lambda_{mn} X_{im} X_{in} - \sum_{im} \mu_m X_{im} \right] = 0, \end{aligned} \quad (22)$$

so that $2 \sum_j L_{kj} X_{jl} - 2 \sum_m X_{km} \lambda_{ml} - \mu_l = 0$, or in matrix notation $\mathbf{L} \mathbf{X} = \mathbf{X} \mathbf{\Lambda} + \frac{1}{2} \mathbf{1} \boldsymbol{\mu}^T$, where $\mathbf{\Lambda}$ is a $(k-1) \times (k-1)$ symmetric matrix of Lagrange multipliers and $\boldsymbol{\mu}$ is a $(k-1)$ -dimensional vector. As before, we can multiply on the left by $\mathbf{1}^T$ to show that $\boldsymbol{\mu} = 0$, and hence we find that

$$\mathbf{L} \mathbf{X} = \mathbf{X} \mathbf{\Lambda}. \quad (23)$$

For a matrix \mathbf{X} satisfying this equation the relaxed cut size R_x , Eq. (19), is

$$R_x = \frac{1}{2} \text{Tr}(\mathbf{X}^T \mathbf{L} \mathbf{X}) = \frac{1}{2} \text{Tr}(\mathbf{X}^T \mathbf{X} \mathbf{\Lambda}) = \frac{n}{2k} \text{Tr} \mathbf{\Lambda}, \quad (24)$$

where we have used Eq. (21).

However, neither the remaining Lagrange multiplier matrix $\mathbf{\Lambda}$ nor the matrix \mathbf{X} itself is completely determined by Eq. (23) and the conditions (20) and (21), and

for good reason. Suppose we have a correct solution for \mathbf{X} and a corresponding $\mathbf{\Lambda}$ such that Eqs. (20) and (21) are satisfied and (19) is minimized. Then consider another matrix $\mathbf{X}' = \mathbf{X} \mathbf{Q}$ where \mathbf{Q} is any $(k-1) \times (k-1)$ orthogonal matrix. In that case \mathbf{X}' also satisfies (20) and (21):

$$\mathbf{1}^T \mathbf{X}' = \mathbf{1}^T \mathbf{X} \mathbf{Q} = 0, \quad (25)$$

$$\mathbf{X}'^T \mathbf{X}' = \mathbf{Q}^T \mathbf{X}^T \mathbf{X} \mathbf{Q} = \frac{n}{k} \mathbf{Q}^T \mathbf{Q} = \frac{n}{k} \mathbf{I}, \quad (26)$$

and substituting $\mathbf{X} = \mathbf{X}' \mathbf{Q}^T$ into Eq. (23) gives $\mathbf{L} \mathbf{X}' \mathbf{Q}^T = \mathbf{X}' \mathbf{Q}^T \mathbf{\Lambda}$, or equivalently

$$\mathbf{L} \mathbf{X}' = \mathbf{X}' \mathbf{Q}^T \mathbf{\Lambda} \mathbf{Q}. \quad (27)$$

In other words, if a solution \mathbf{X} exists for Eq. (23), then any rotation \mathbf{X}' is also a solution of the same equation but with transformed Lagrange multipliers $\mathbf{\Lambda}' = \mathbf{Q}^T \mathbf{\Lambda} \mathbf{Q}$. But $\mathbf{\Lambda}'$ will have the same trace as $\mathbf{\Lambda}$, since the trace is invariant under an orthogonal transformation of this kind, and hence the value of R_x , Eq. (24), is unchanged, meaning this new solution also minimizes R_x . Thus any rotation of a minimum of R_x is also a minimum. This is natural—only the geometry of the vertices relative to one another matters and not their absolute positions. The same is also true of the original simplex vectors, which can be freely rotated without affecting the cut size.

What this means in practice is that we are at liberty to choose any rotation of the solution we prefer, whichever is most convenient. All will give equally good values for the relaxed cut size R_x . But when we “unrelax” the solution again to get back to the original simplex vectors we must bear in mind that different rotations do not necessarily give the same unrelaxed solution—some rotations may be better than others. This means that the unrelaxation process may require an additional rotation to find the best solution, an extra step in the algorithm that we will perform using a polar decomposition.

For our purposes the most convenient rotation of the solution is the one for which $\mathbf{Q}^T \mathbf{\Lambda} \mathbf{Q}$ is diagonal. Such a rotation always exists since $\mathbf{\Lambda}$ is real and symmetric. For this choice Eq. (27) takes the form of Eq. (23) again but with $\mathbf{\Lambda}$ diagonal, which means that each column of \mathbf{X} is an eigenvector of the graph Laplacian with the diagonal elements of $\mathbf{\Lambda}$ giving the eigenvalues. Then the conditions (20) and (21) tell us that the eigenvectors must be distinct (because they are orthogonal), identically normalized, and orthogonal to the vector $\mathbf{1}$. Equation (24) tells us that to minimize R_x we should choose the eigenvalues as small as possible and, given that we are forbidden by Eq. (20) from choosing the lowest (zero) eigenvalue, because its eigenvector is the vector $\mathbf{1}$, our best choice is to choose the columns of \mathbf{X} to be the eigenvectors corresponding to the second- to k th-lowest eigenvalues of the Laplacian. Which column is which is arbitrary and unimportant, again corresponding only to a rotation—any choice will work equally well.

C. Reversing the relaxation

The developments of the previous section give us an exact solution to the relaxed optimization problem. The final step in the algorithm is the reversal of the relaxation process to recover an approximation to the ideal partitioning of the graph. In principle this is, as with the two-group case, a straightforward operation—we round each row of the matrix \mathbf{X} to the nearest simplex vector. More specifically, we write the i th row \mathbf{x}_i of the matrix as a linear combination of the simplex vectors \mathbf{w}_j :

$$\mathbf{x}_i = \sum_j (\mathbf{w}_j^T \mathbf{x}_i) \mathbf{w}_j. \quad (28)$$

(This looks like a standard orthogonal decomposition, but it's not because the simplex vectors are not orthogonal. Nonetheless it is straightforward to prove that such a decomposition always exists when the vectors \mathbf{w}_j point to the corners of a regular simplex.) In rounding the points $\mathbf{x}_i \rightarrow \mathbf{w}_j$ we choose that simplex vector \mathbf{w}_j whose contribution, equal to the inner product $\mathbf{w}_j^T \mathbf{x}_i$, is largest in Eq. (28). An equivalent statement is that we simply round the points to the nearest corner of the simplex, defined in terms of ordinary Euclidean distance in the $(k-1)$ -dimensional space. As in the two-group case, this is merely a heuristic—there is no guarantee that it will give an optimal partitioning, although as we will see it appears to work well in practice.

This, however, is not the whole story. As we mentioned in the previous section, there is a complication: both the rows of \mathbf{X} and the simplex vectors are only fixed up to a rotation, so an additional rotation may be required before rounding to find the best solution. We can rotate either the rows of \mathbf{X} or the simplex vectors. We recommend rotating the simplex vectors, since there are only k of them, whereas there are n rows of the matrix \mathbf{X} .

The situation is depicted in Fig. 4 for the case $k=3$. The rows \mathbf{x}_i are two-dimensional vectors in this case and form a scatter of points in the plane of the plot as shown. The points do indeed approximate reasonably well to the corners of a simplex (an equilateral triangle in this case), so in principle we should be able to round them off and get a good solution to the partitioning problem. But we don't know *a priori* what the correct orientation of the simplex is, and in this case our first guess, shown as the Y shape in the figure, is off and a rotation is required. Since our goal is to round the \mathbf{x}_i to simplex vectors \mathbf{w}_j such that the inner products $\mathbf{w}_j^T \mathbf{x}_i$ are large, a sensible criterion for selecting the best rotation (there are others) is to maximize the sum of the inner products for all vertices.

Given an assignment of vertices to groups, we can write down the matrix \mathbf{S} of (unrotated) simplex vectors, and the rotated vectors take the form $\mathbf{S}\mathbf{Q}$, where \mathbf{Q} is a $(k-1) \times (k-1)$ orthogonal matrix as before. Then the sum of the inner products of the rotated vectors with the \mathbf{x}_i is $\text{Tr}(\mathbf{Q}^T \mathbf{S}^T \mathbf{X})$ and the task of finding the best rotation is equivalent to maximizing this trace over all possible

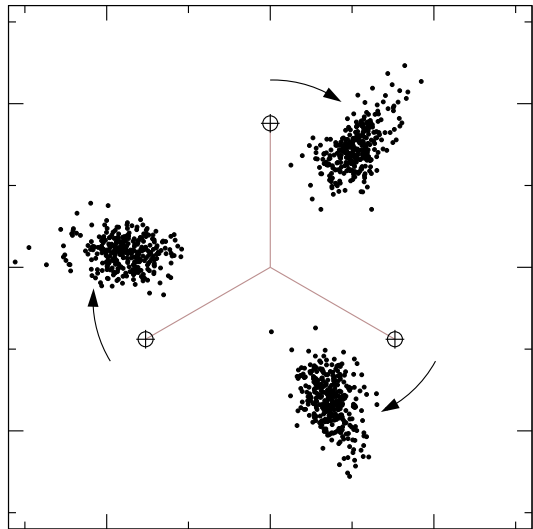


FIG. 4: The points in this plot represent the elements of the second and third eigenvectors of the Laplacian for a small graph of about a thousand vertices. The particular graph used here was, for the purposes of illustration, deliberately created with three communities in it, using a simple planted partition model [17] in which edges are placed between vertices independently at random, but with higher probability between vertices in the same group than between those in different groups. As the figure shows, there is a clear clustering of the resulting points into three groups, which fall, roughly speaking, at the corners of a two-dimensional regular simplex, i.e., an equilateral triangle. To determine the division of the graph into groups, we need to round these points to the nearest simplex vector (the three ends of the Y shape), but the simplex must first be rotated to match the orientation of the points.

orthogonal matrices \mathbf{Q} . This maximization is a standard problem in so-called Procrustes analysis [18]. It can be solved by performing a singular value decomposition of the matrix $\mathbf{S}^T \mathbf{X}$:

$$\mathbf{S}^T \mathbf{X} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T, \quad (29)$$

where $\mathbf{\Sigma}$ is the diagonal matrix of singular values and \mathbf{U} and \mathbf{V} are orthogonal matrices. Then

$$\begin{aligned} \text{Tr}(\mathbf{Q}^T \mathbf{S}^T \mathbf{X}) &= \text{Tr}(\mathbf{Q}^T \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T) = \text{Tr}(\mathbf{V}^T \mathbf{Q}^T \mathbf{U} \mathbf{\Sigma}) \\ &\leq \text{Tr} \mathbf{\Sigma}, \end{aligned} \quad (30)$$

the inequality following because $\mathbf{V}^T \mathbf{Q}^T \mathbf{U}$, being a product of orthogonal matrices, is also itself orthogonal, and all elements of an orthogonal matrix are less than or equal to 1. It is now trivial to see that the exact equality—which is, by definition, the maximum of $\text{Tr}(\mathbf{Q}^T \mathbf{S}^T \mathbf{X})$ with respect to \mathbf{Q} —is achieved when $\mathbf{Q}^T = \mathbf{V} \mathbf{U}^T$ or equivalently when

$$\mathbf{Q} = \mathbf{U} \mathbf{V}^T. \quad (31)$$

The product $\mathbf{U} \mathbf{V}^T$ is the orthogonal part of the *polar decomposition* of $\mathbf{S}^T \mathbf{X}$. Calculating it in practice in-

volves calculating first the singular value decomposition, Eq. (29), and then discarding the diagonal matrix Σ . Note that $\mathbf{S}^T \mathbf{X}$ is only a $(k-1) \times (k-1)$ matrix (not an $n \times n$ matrix), and hence its singular value decomposition can be calculated rapidly provided k is small, in $O(k^3)$ time.

These developments assume that we know the assignment of the vertices to the groups. In practice, however, we don't. (If we did, we wouldn't need to partition the graph in the first place.) So in the algorithm we propose we start with a random guess at the orientation of the simplex. We round the rows of \mathbf{X} to the simplex vectors to determine group memberships and then rotate the simplex vectors to fit the resulting groups according to Eq. (31). We repeat this procedure until the groups no longer change. In a clear-cut case like that of Fig. 4, only one or two iterations would be needed for convergence, but in more ambiguous cases we have found that as many as half a dozen or more iterations may be necessary.

As in the two-group case, we are not guaranteed that the groups found using this method will be exactly equal in size. The relaxed optimization must satisfy Eq. (20), but the corresponding condition, Eq. (17), for the unrelaxed division of the graph is typically only satisfied approximately and hence the groups will only be approximately equal in size. As in the two-group case, however, this is typically not a problem. Often we are content with nearly equal groups, but if not the groups can be balanced using a post-processing step. For example, the division into equally sized groups that maximizes the sum of the inner products $\mathbf{w}_j^T \mathbf{x}_i$ can be calculated exactly in polynomial time using the so-called Hungarian algorithm [19], or approximately using a variety of vertex moving heuristics.

D. Summary of the algorithm and running time

Although the derivation of the previous sections is moderately lengthy, the final algorithm is straightforward. In summary the algorithm for partitioning a given graph into k groups is as follows.

1. Compute the $n \times (k-1)$ matrix \mathbf{X} whose columns consist of the eigenvectors of the graph Laplacian corresponding to the second- to k th-lowest eigenvalues.
2. Choose at random an initial orientation of the $(k-1)$ -dimensional regular simplex and construct the $k \times (k-1)$ matrix \mathbf{W} whose rows are the vectors \mathbf{w}_j that point to the corners of the simplex.
3. Compute the inner products $\mathbf{w}_j^T \mathbf{x}_i$ for all vertices i and groups j , then assign each vertex i to the group j for which the product is largest.
4. Using the resulting assignment of vertices to groups, compute the matrix \mathbf{S} whose i th row is the simplex vector \mathbf{w}_j for the group j to which vertex i

belongs, then compute the matrix of rotated simplex vectors $\mathbf{W}' = \mathbf{W}\mathbf{U}\mathbf{V}^T$, where $\mathbf{U}\Sigma\mathbf{V}^T$ is the singular value decomposition of the $(k-1) \times (k-1)$ matrix $\mathbf{S}^T \mathbf{X}$.

5. Replace \mathbf{W} by \mathbf{W}' and repeat from step 3 until the group memberships no longer change.

Because the initial orientation of the simplex is random, the results of the algorithm are not deterministic, and hence, as with other randomized algorithms, it may be beneficial to run the calculation several times with different initial orientations and among the divisions of the graph so generated choose the one that has the smallest cut size. A similar procedure is typically used, for example, for partitioning with k -means, which also employs a random initial condition. (Note that the computation of the eigenvectors—step 1 above—does not have to be repeated; only the randomized part of the algorithm, from step 2 onward, need be repeated.)

In practice the algorithm runs quickly. Most often we are interested in sparse graphs in which the number of edges is proportional to the number of vertices, so that the mean degree of a vertex tends to a constant as the graph becomes large. In this situation the eigenvectors of the Laplacian can be calculated using sparse methods such as the Lanczos algorithm in time $O(k^2 n)$. The other steps of the algorithm all also take time $O(k^2 n)$, except for the singular value decomposition, which takes time $O(k^3)$, and hence (since $k < n$) one iteration of the algorithm has leading-order worst-case running time $O(k^2 n)$. We do not have a rigorous understanding of how the number of iterations will scale with n and k but the number appears in practice to be small—less than about ten iterations for the graphs we study—and to increase only slowly, if at all, with graph size. We conjecture, therefore, that the running time of the algorithm scales approximately as $k^2 n$, making it well suited for large graphs.

E. Weighted graphs and data clustering

The methods described in the previous sections can be extended in a straightforward manner to weighted graphs—graphs with edges of varying strength represented by varying elements in the adjacency matrix. For such graphs the goal of partitioning is to divide the vertices into groups such that the sum of the weights of the edges running between groups is minimized. To achieve this we generalize the degree d_i of vertex i in the obvious fashion $d_i = \sum_j A_{ij}$ and the elements of the Laplacian accordingly $L_{ij} = d_i \delta_{ij} - A_{ij}$. Then the cut size once again satisfies Eq. (16), and the rest of the algorithm follows as before. We have not experimented extensively with applications to weighted graphs, but in preliminary tests the results look promising.

One can also apply our methods to the problem of data clustering, the grouping of points within a multidimen-

sional data space into clusters of similar values [4, 14]. One standard approach to this problem makes use of an affinity matrix. Suppose one has a set of n points represented by vectors \mathbf{r}_i in a d -dimensional data space. One then defines the affinity matrix \mathbf{A} to have elements

$$A_{ij} = e^{-|\mathbf{r}_i - \mathbf{r}_j|^2 / 2\sigma^2}, \quad (32)$$

where σ is a free parameter chosen by the user. If σ is roughly of order the distance between intra-cluster points, then A_{ij} will approximately take the form of the adjacency matrix of a weighted graph in which vertices are connected by strong edges if the corresponding data points are near neighbors in the data space. (For values of σ much larger or smaller than this clustering methods based on the affinity matrix will not work well, so some care in choosing σ is necessary to get good results. Automated methods have been proposed for choosing a good value [10].)

Given the affinity matrix, we can now apply the method described above for weighted graphs to this matrix and derive a clustering of the data points. We will not pursue this idea further in the present paper, but in preliminary experiments on standard benchmark data sets we have found that the algorithm gives results comparable with, and in some cases better than, other spectral clustering methods.

IV. RESULTS

Our primary purpose in this paper is to provide a first-principles derivation of a spectral partitioning algorithm. However, given that the algorithm we have derived differs from standard algorithms, it is also interesting to examine how well it performs in practice. In this section we give a number of example applications of the algorithm to graphs from a variety of sources. Our tests do not amount to an exhaustive characterization of performance, but they do give a good idea of the basic picture. Overall, we find that the algorithm has performance comparable to that of other spectral algorithms based on Laplacian eigenvectors, but there exist classes of graphs for which our algorithm does measurably better. In particular, the algorithm appears to perform better than some competitors in cases where the partitioning task is particularly difficult.

As a first example, Fig. 5 shows the result of applying our algorithm to a graph from the widely used University of Florida Sparse Matrix Collection. This graph is a two-dimensional mesh network drawn from a NASA structural engineering computation, and is typical of finite-element meshes used in such calculations (which are a primary application of partitioning methods). Figure 5 shows a split of the graph into four parts. The split is closely similar to that found by conventional spectral partitioning using k -means, indicating that our algorithm has comparable performance on this application to standard methods in current use.

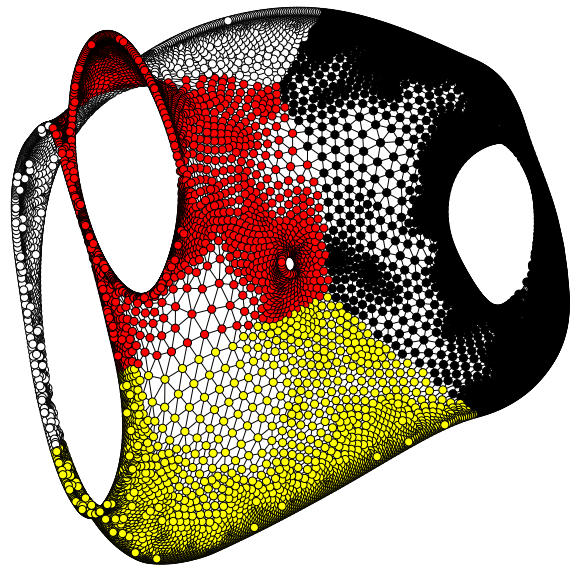


FIG. 5: Division of a structural engineering mesh network of 15 606 vertices into four parts—represented by the four colors—using the algorithm described in this paper. The complete graph has 45 878 edges; this division cuts just 362 of them, less than 1%. Graph data courtesy of the University of Florida Sparse Matrix Collection.

Figure 6 shows an application to a graph representing a power grid, specifically the Western States Power Grid, which is the network of high-voltage electricity transmission lines that serves the western part of the United States [20]. The figure shows the result of splitting the graph into four parts and the split is an intuitively sensible one and again comparable to that found using more traditional methods.

There are, however, some graphs for which our method gives results that are measurably, if only modestly, better than those given by previous methods. As a controlled test of the performance of the algorithm we have applied it to artificial graphs generated using a planted partition model [17] (also called a stochastic block model in the statistical literature [21]). In this model one creates graphs with known partitions by dividing a specified number of vertices into groups and then placing edges within and between those groups independently with given probabilities. In our tests we generated graphs of 900 vertices with three equally sized groups of 300 vertices each. Edges were placed between vertices with two different probabilities, one for vertices in the same group and one for vertices in different groups, chosen so that the average degree of a vertex remained constant at 30. We then varied the ratio of in-group to between-group connections to test the performance of the algorithm.

Figure 7 shows the results of applying both our algorithm and the standard k -means spectral algorithm to a large set of graphs generated using this method. We compared the divisions found by each algorithm to the

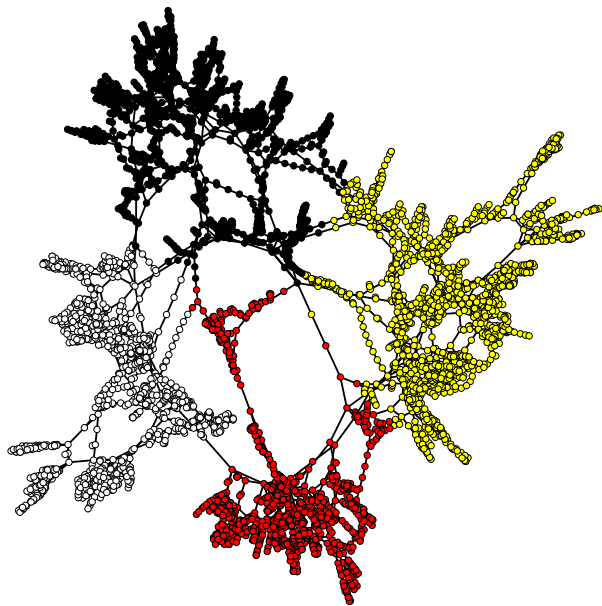


FIG. 6: Division into four parts of a graph representing the Western States Power Grid of the United States. The complete graph contains 4941 vertices and 6594 edges, of which 33 are cut in this division. Graph data courtesy of Duncan Watts.

known correct divisions and calculated the fraction of vertices classified into the correct groups as a function of the average number c_{out} of edges from a vertex to vertices in other groups. For small values of c_{out} the groups in the graph should be clear and we expect any algorithm to do a good job of finding the best cut. But as c_{out} is increased the problem becomes more challenging and the fraction of correct vertices declines for both algorithms, eventually approaching the value $\frac{1}{3}$, denoted by the horizontal dashed line in the figure, at which classification is no better than chance—we would expect a random division of vertices into groups to get a third of the vertices right just by luck. Just before we reach this point, however, in the regime corresponding to the hardest partitioning problems that can still be solved by these algorithms, we see that our algorithm does measurably better than k -means, and the difference, while small, is statistically significant given the errors on the measurements. In this regime, therefore, in the context of this particular test, it appears that our algorithm returns the better results.

To be fair, we should also point out that there are some cases in which the k -means algorithm outperforms the algorithm of this paper. In particular, we find that for graphs generated using the planted partition model, again with three groups, but where the between-group connections are asymmetric and one pair of groups is more weakly connected than the other two pairs, the k -means algorithm does better in certain parameter regimes. The explanation for this phenomenon appears to be that our algorithm has difficulty finding the best

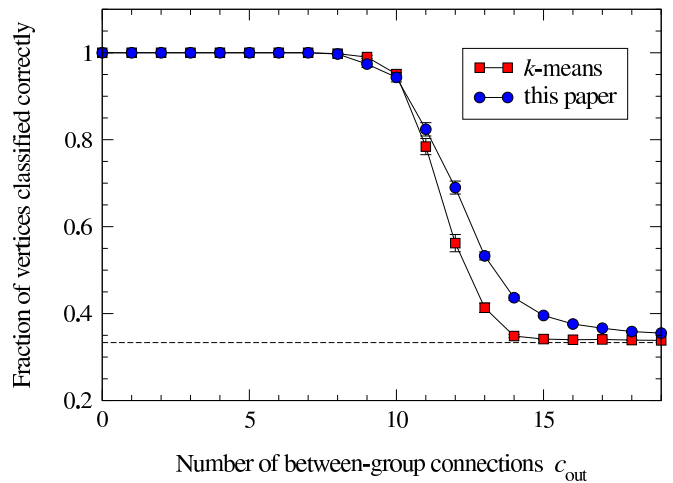


FIG. 7: Fraction of vertices classified into the correct groups by a standard spectral algorithm based on k -means and by the algorithm described in this paper, when applied to graphs artificially generated using a planted partition model. The dashed horizontal line represents the point at which the algorithms do no better than chance. For both algorithms we use a random initial condition as described in the text and the results reported are the best of five runs.

orientation of the simplex to perform the partitioning. It is possible that one could achieve better results using a different method for finding the orientation. The k -means partitioning algorithm, which does not use an orientation step, has no corresponding issues.

V. CONCLUSIONS

In this paper, we have derived a spectral partitioning algorithm from first principles, as a relaxation approximation to a well-defined minimum-cut problem. This contrasts with more traditional presentations in which an algorithm is proposed *ex nihilo* and then proved after the fact to give good results. While both approaches have merit, ours offers an alternative viewpoint that helps explain why spectral algorithms work—because the spectral algorithm is, in a specific sense, an approximation to the problem of minimizing the cut size over balanced divisions of the graph.

Our approach not only offers a new derivation, however; the end product, the algorithm itself, is also different from previous algorithms, involving an additional rotation step that is performed using a polar decomposition. In practice, the algorithm appears to give results that are comparable with those of previous algorithms and in some cases measurably, if only modestly, better. The algorithm is also efficient. For graphs of n vertices divided into k groups, the running time is $O(k^2n)$ for a single iteration of the algorithm. Repeated iterations are needed to reach the final answer but empirically the

number of iterations appears to grow only slowly with graph size, so overall running time is roughly linear in n for given k .

The developments described here leave some questions unanswered. We have, for example, treated only the case of equal group sizes and not addressed how the algorithm might be generalized to the unequal case, although this does not appear to present any great challenges—unequal sizes merely add a constant to the positions of points in the spectral embedding, but the rest of the calculation follows through much as before. Another variant of the problem arises when the group sizes are not exactly equal but are allowed to vary within limits. This type of problem could be addressed using the algorithm described in

this paper but with an additional post-processing step that moves vertices in order to bring the sizes within the required bounds. These and related ideas we leave for future work.

Acknowledgments

The authors thank Raj Rao Nadakuditi for useful insights and suggestions. This work was funded in part by the National Science Foundation under grant DMS-1107796.

-
- [1] U. Elsner, Graph partitioning—a survey. Technical Report 97-27, Technische Universität Chemnitz (1997).
 - [2] P.-O. Fjällström, Algorithms for graph partitioning: A survey. *Linköping Electronic Articles in Computer and Information Science* **3**(10) (1998).
 - [3] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco (1979).
 - [4] U. von Luxburg, A tutorial on spectral clustering. *Statistics and Computing* **17**, 395–416 (2007).
 - [5] M. Fiedler, Algebraic connectivity of graphs. *Czech. Math. J.* **23**, 298–305 (1973).
 - [6] A. Pothen, H. Simon, and K.-P. Liou, Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.* **11**, 430–452 (1990).
 - [7] C. J. Alpert and S.-Z. Yao, Spectral partitioning: The more eigenvectors, the better. In B. T. Preas, P. G. Karger, B. S. Nobandegani, and M. Pedram (eds.), *Proceedings of the 32nd International Conference on Design Automation*, pp. 195–200, Association of Computing Machinery, New York, NY (1995).
 - [8] J. Shi and J. Malik, Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22**, 888–905 (2000).
 - [9] M. Meilă and J. Shi, Learning segmentation by random walks. In T. K. Leen, T. G. Dietterich, and V. Tresp (eds.), *Proceedings of the 2000 Conference on Advances in Neural Information Processing Systems*, MIT Press, Cambridge, MA (2000).
 - [10] A. Y. Ng, M. I. Jordan, and Y. Weiss, On spectral clustering: Analysis and an algorithm. In T. G. Dietterich, S. Becker, and Z. Ghahramani (eds.), *Proceedings of the 2001 Conference on Advances in Neural Information Processing Systems*, MIT Press, Cambridge, MA (2001).
 - [11] M. Meilă and L. Xu, Multiway cuts and spectral clustering. Technical report, University of Washington, Department of Statistics (2003).
 - [12] R. Kannan, S. Vempala, and A. Vetta, On clusterings: Good, bad and spectral. *J. ACM* **51**, 497–515 (2004).
 - [13] J. R. Lee, S. O. Gharan, and L. Trevisan, Multi-way spectral partitioning and higher-order Cheeger inequalities. Preprint arXiv:1111.1055 (2011).
 - [14] D. Verma and M. Meilă, Comparison of spectral clustering methods. Technical Report CSE-03-05-01, University of Washington (2003).
 - [15] F. Bach and M. I. Jordan, Learning spectral clustering, with application to speech separation. *Journal of Machine Learning Research* **7**, 1963–2001 (2006).
 - [16] Z. Zhang and M. I. Jordan, Multiway spectral clustering: A margin-based perspective. *Statistical Science* **23**, 383–403 (2008).
 - [17] A. Condon and R. M. Karp, Algorithms for graph partitioning on the planted partition model. *Random Structures and Algorithms* **18**, 116–140 (2001).
 - [18] J. C. Gower and G. B. Dijkstra, *Procrustes Problems*. Oxford University Press, Oxford (2004).
 - [19] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Dover, New York (1998).
 - [20] D. J. Watts and S. H. Strogatz, Collective dynamics of ‘small-world’ networks. *Nature* **393**, 440–442 (1998).
 - [21] T. A. B. Snijders and K. Nowicki, Estimation and prediction for stochastic blockmodels for graphs with latent block structure. *Journal of Classification* **14**, 75–100 (1997).